Touch Pattern Recognition for authentication

13.05.2023

—

Diarra Yacouba, Kanté Younouss Koly, Haïdara Mohamed,

Diabaté Maïmouna, Maïga Maïmouna Henna

@Institut Privée Africain de Technologie et de Management (ITMA)

Baco Djicoroni ACI, Bamako, Mali

Baco Djicoroni ACI, Bamako, Mali

Overview

This report aims to be a good checkpoint about the progression of the Touch Pattern Recognition research project.
The Touch Pattern Recognition project is a project that aspires to explore new ways to authentication systems. This project comes from five students at the @ITMA university Diarra Yacouba, Kanté Younouss Koly, Haïdara Mohamed, Diabaté Maïmouna and Maïga Maïmouna Henna as part of the second edition of "TRANSFORMATION NUMERIQUE", the competition organized by Africa_Digital in May 2023.
These students participated in the "TRANSFORMATION NUMERIQUE" universities league competing about Data security.
Note that this file can be edited at any moment due to the ongoing status of the project and that all events mentioned in this document are localized in Bamako/Mali between April 30 and July 13 2023.

With that said, The idea behind the project is to replace traditional system authentication credentials (username/ password) by a Deep learning (DL) model trained to Recognize the user of the system through touch data on touch sensitive devices.
For this purpose data are represented with respect to different fields such as positions, speeds, directions, latency (delay) of each touch.

This document is going to talk about the work done by the previously mentioned students, from data acquisition to Model deployment passing through issues and problems encountered.

Goals

This project aims to:

1. Rethink the way we think of authentication systems.
2. Bring more innovation and security into authentication systems.
3. Make access to systems easier by replacing passwords that can be tedious stuff as they are easy to forget or hack.
4. Look for new applications of AI systems that are becoming more and more powerful.

Data acquisition & preprocessing

We are going to talk about the way data has been collected and processed for training.

I. Data Acquisition

Firstly, we utilized an existing mobile web application called "TouchTracker" developed by Marco Cancellieri. This application has a simple user interface that displays a green screen and records all gestures performed on the screen written in JavaScript.
More info at : TouchTracker

However, the application's original purpose was not to collect touch-related data that we need, so we made modifications to its source code to collect the necessary data. In Particular, we edited the application's code to collect touch positions, speeds, directions, durations, and timestamps.

These modifications allowed us to gather detailed information about the touch gestures being performed on the screen, which is essential for training a machine learning/Deep learning model to recognize and make predictions accurately.

Once the data has been collected, it will be exported in .json files which is a structured data format that is easily readable and processed by most programming languages, including Python, the language we used to train your neural nets.

To take up with the data acquisition process, we created a user interface for our TouchTracker based web app, adding some HTML, CSS, and JS code.

HTML, CSS, and JS are the three foundational technologies used to create websites and web applications. HTML provides the structure of the page, CSS controls the presentation and layout, and JS handles the interactivity and functionality.

Note: This report may be read by people with no experience of the fields our work touched to, that's why we are introducing technologies like HTML that are well know from IT's fellas

We needed to modify the JS code to handle touch events. This likely involved rewriting or editing event listeners to recognize touch gestures such as taps, swipes, and pinches, and to collect the touch-related data we needed for the training.

Overall, our data acquisition process involved modifying existing mobile web applications, collecting detailed touch-related data, and exporting the data in a compatible format.

Note that at the date this report was written the application was not completely finished due to issues that will be described in the last section of this document.

II. Preprocessing

After data acquisition we had to represent data in a suitable form for training an artificial neural net (You can think of those like an abstraction of human brain neural networks for high level computing). For this purpose we wrote a python script that read each downloaded .json files and extract fields we was interested in such as positions (list of 2 items' lists where the first is Position.X and the second Position.Y with respect to a fictitious reference point on the screen), speeds (list of Euclidean speeds calculated for each new position recorded with respect to the touch's starting point), directions (list of direction/angle of each touch with regard to the previous in radians) and timestamps that we used to compute the duration of each touch.
Note that positions have been split into two new lists position.X and position.Y.

And then in order to obtain the features that will be used to train the model we calculated the mean, standard derivation, min, max and range of each field or lists (Position.X, Position.Y, speeds, directions) and appended the duration of the touch. This way we got a new representation of our data that is a numpy.array of shape (21,) which is a pretty good data representation for training a Deep Learning model.

Data has then been labeled and saved with **Pyhton.pickle** .

NB: Label was binary values (0, 1)

AI Training & Evaluation

I. Creation & Training

We decided to go for a Keras model.

Keras is a Python library for deep learning that wraps the efficient numerical libraries Tensorflow and Theano. Keras allows you to quickly and simply design and train neural networks and deep learning models.

So we create a quite simple Keras Sequential model.

The weights are initialized using a small Gaussian random number. The Rectifier activation function is used. The output layer contains a single neuron in order to make predictions. It uses the sigmoid activation function in order to produce a

probability output in the range of 0 to 1 that can easily and automatically be converted to crisp class values.

Finally, we used the logarithmic loss function (cross-entropy) during training, one of the most effective loss function for binary classification problems. The model also uses the efficient Adam optimizer for gradient descent, and accuracy metrics will be calculated during the training.

The model had just one hidden layer of 10 neurons. So the Model topology can be summarized by [21, 10, 1]. It had just 693 parameters

II. Evaluation and saving

We firstly evaluated couple model topologies using stratified cross validation from the scikit-learn framework.
The idea was to use this step as a test consisting of training for a short while and evaluate them in order to choose the best model to train for a longer time.

The stratified cross validation (StratifiedKFold) is a resampling technique that will provide an estimate of the performance of the model. It does this by splitting the data into k-parts and training the model on all parts except one, which is held out as a test set to evaluate the performance of the model.

We have been really surprised by the performance of the model even without any hidden layer that scored 83% accuracy. That, one more time, comes up with the question of data quality. Haven't we been training with biased data?

After further training of just 1000 epochs the topology [21, 10, 1] scored up to 90% accuracy and it's the model we have saved at the time of writing this document (May 13, 2023).

We then saved the pre-trained models ready for inference, test or further training in different formats:

- In a folder

The SavedModel format is a directory containing a protobuf binary and a TensorFlow checkpoint. The first approach used to save the model

- In a TF.js Layers format

We saved the model in TF.js Layers format after a brief test to use it with the app later

Model Deployment

We already exported the model to Tensorflow.js layers format and integrated it with success to the web application used to collect data. Although some issues remains like the model's precision decreasing once deployed or the inference on iphones we have being able to test our trained model on androids with no very good results

Issues

In this section we are going to talk about two issues or tasks that have not been corrected yet.

I. Improving the UI of the app

We would like to improve the aesthetic of the app and eventually add some features to make the user experience better when testing the model, even though the web app isn't essential to the project.

II. Correct bugs on iphones

We should solve the problems related to inferences on iphones soon or come up with an app designed especially for iphones.

Summary & Notes

Summarizing this document was presenting a little research project idea explored by five students in Mali.

With results described here we think that this project has a big potential and would like to get people interested in it. But we can't state the effectiveness, affordability and reliability of such systems if they were really complete.

There is nothing spectacular for now but we hope it will be. Even though we will probably stop our work on this project for now, we hope ourselves and/or others coming up with new ideas to improve the core idea of the project and the work done

You can get all codes written to this date on this github and contacts authors with the following links:

Kanté Younouss Koly: email

Haïdara Mohamed: email, Facebook, linkedin,

Diabaté Maïmouna: email

Diarra Yacouba: email, Facebook, linkedin, twitter, github, Stackoverflow

Maïga Maïmouna Henna: email, twitter, linkedin